Sensing, Navigation and Reasoning Technologies for the DARPA Urban Challenge

Mohammed Aly Joel W. Burdick Vanessa Carson Stefano Di Cairano
Noel duToit Melvin Flores Jessica Gonzalez Andrew Howard
Laura Lindzey Jeremy Ma Richard M. Murray[2] Richard Petras
Sam Pfister Dominic Rizzo Tichakorn Wongpiromsarn

California Institute of Technology/Jet Propulsion Laboratory

Team Caltech
13 April 2007

Executive Summary

This paper describes Team Caltech's technical approach and current progress towards developing new technologies to enable it to compete in and win the 2007 DARPA Urban Challenge. Our primary technical thrusts are are three areas: (1) mission and contingency management for autonomous systems; (2) distributed sensor fusion, mapping and situational awareness; and (3) optimization-based guidance, navigation and control. Preliminary work in each of these areas, combined with a systematic approach to overall systems engineering and field testing, has demonstrated the ability for our system to navigate in traffic situations consistent with the DARPA Technical Criteria. The development of an optimization-based, dynamic planner has proceeded more slowly than expected and has held back our progress in accomplishing basic traffic and advanced navigation tasks. System level tests have been used to identify additional areas for continued work over the remaining 203 days until the competition.

1 Introduction and Overview

Team Caltech was formed in February of 2003 with the goal of designing a vehicle that could compete in the 2004 DARPA Grand Challenge. Our 2004 vehicle, Bob, completed the qualification course and traveled approximately 1.3 miles of the 142-mile 2004 course. In 2004-05, Team Caltech developed a new vehicle, Alice, to participate in the 2005 DARPA Grand Challenge. Alice utilized a highly networked control system architecture to provide high performance, autonomous driving in unknown environments. The system successfully completed several runs in the National Qualifying Event, but encountered a combination of sensing and control issues in the Grand Challenge Event that led to a critical failure after traversing approximately 8 miles.

As part of the 2007 Urban Challenge, Team Caltech is developing new technology for Alice in three key areas: (1) mission and contingency management for autonomous systems; (2) distributed sensor fusion, mapping and situational awareness; and (3) optimization-based guidance,

[2]Corresponding author: murray@cds.caltech.edu

navigation and control. This section provides a summary of the capabilities of our existing vehicle, and describes the framework that we are using to create a robust and reliable vehicle capable of winning the 2007 Urban Challenge.

Problem Description and Approach For the 2007 Urban Challenge, we are building on the basic architecture that was deployed by Caltech in the 2005 race, but providing significant extensions and major additions that allow operation in the more complicated (and uncertain) urban driving environment. Our primary approach in the desert competition was to construct an elevation map of the terrain sounding the vehicle and then convert this map into a cost function that could be used to plan a high speed path through the environment. A supervisory controller provided contingency management by identifying selected situations (such as loss of GPS or lack of forward progress) and implementing tactics to overcome these situations.

For the urban challenge, several new challenges must be addressed. Road location must be determined based on lane and road features, static and moving obstacles must be avoided, and intersections must be successfully navigated. We have again chosen a deliberative planning architecture, in which a representation of the environment is built up through sensor data and motion planning is done using this representation. A significant issue is the need to reason about traffic situations in which we interact with other vehicles or have inconsistent data about the local environment or traffic state.

System Architecture A key element of our system is the use of a networked control systems (NCS) architecture that we developed in the first two grand challenge competitions. Building on the open source *Spread* group communications protocol, we have developed a modular software architecture that provides inter-computer communications between sets of linked processes [2]. This approach allows the use of significant amounts of distributed computing for sensor processing and optimization-based planning, as well as providing a very flexible backbone for building autonomous systems and fault tolerant computing systems. This architecture also allows us to include new components in a flexible way, including modules that make use of planning and sensing modules from the Jet Propulsion Laboratory (JPL) and the OTGX software from Northrop Grumman, described in more detail below.

A schematic of the high-level system architecture that we are developing for the Urban Challenge is shown in Figure 1. This architecture shares the same underlying approach as the software used for the 2005 Grand Challenge, but with three new elements:

Canonical Software Architecture for mission and contingency management. The complexity and dynamic nature of the urban driving problem make centralized goal and contingency management impractical. For the navigation functions of our system, we have developed a decentralized approach where each module only communicates with the modules directly above and below it in the hierarchy. Each module is capable of handling the faults in its own domain, and anything the module is unable to handle would be propagated “up the chain” until the correct level had been reached to resolve the fault or conflict. This architecture is described in more detail in Section 2.3 and builds on previous work at JPL [3, 5, 8].

Mapping and Situational Awareness. The sensing subsystem is responsible for maintaining both a detailed geometric model of the vehicle’s environment, as well as a higher level representation of the environment around the vehicle, including knowledge of moving obstacles and road fea-

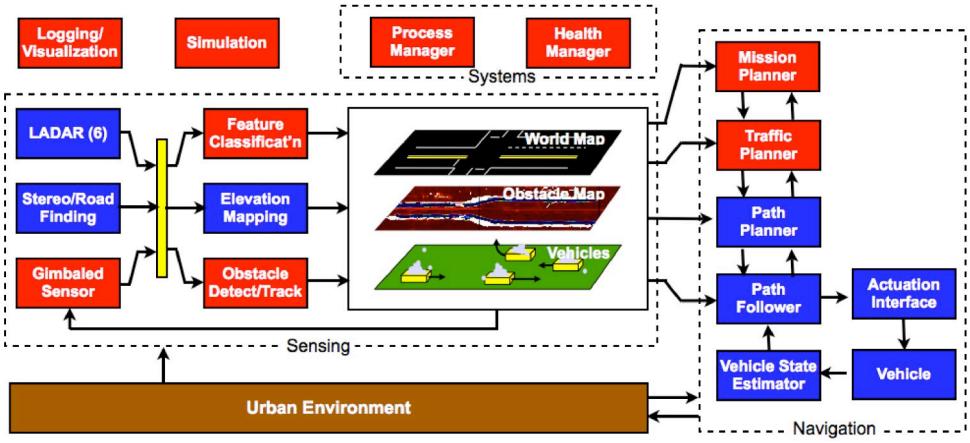

Figure 1: Systems architecture for operation of Alice in the 2007 Challenge. The sensing subsystem is responsible for building a representation of the local environment and passing this to the navigation subsystems, which computes and commands the motion of the vehicle. Additional functionality is provided for process and health management, along with data logging and simulation. The modules in blue (dark shading) were present in the 2005 architecture, the modules in red (lighter shading) are new modules that are currently being tested. Dashed boxes indicate functional subsystems and align with organizational teams.

tures. It associates sensed data with prior information and broadcasts the structure and uncertainty in the environment to the navigation subsystem. The mapping module maintains a vectorized representation of static and dynamic sensed obstacles, as well as detected lane lines, stop lines and waypoints. The map uses a 2.5 dimensional representation where the world is projected into a flat 2D plane, but individual elements may have some non-zero height. Each sensed element is tracked over time and when multiple sensors overlap in field of view, the elements will be fused to improve robustness to false positives as well as overall accuracy. These methods are described in more detail in Section 2.2.

Route, Traffic and Path Planning. The planning subsystem determines desired motion of the system, taking into account the current route network and mission goals, traffic patterns and driving rules, and terrain features (including static obstacles). This subsystem is also responsible for predicting motion of moving obstacles, based on models of driving behavior and traffic rules, and for implementing defensive driving techniques. The planning problem is divided into three subproblems (route, traffic, and path planning) and implemented in separate modules. This decomposition is well-suited to implementation by a large development team and modules can be developed and tested using earlier revisions of the code base as well as using simulation environments described in more detail below. This approach also allows easy access to the different layers of environment representation that are needed by different planning modules. Additional details are provided in Section 2.3.

Design Scenarios Our design choices have been driven through the development of a set of scenarios, derived from the DARPA Technical Criteria, that are used to explore design choices. Three such scenarios are shown in Figure 2, which shows examples of “corridor plans” for moving through an intersection, navigating in a parking lot, and executing a U-turn.

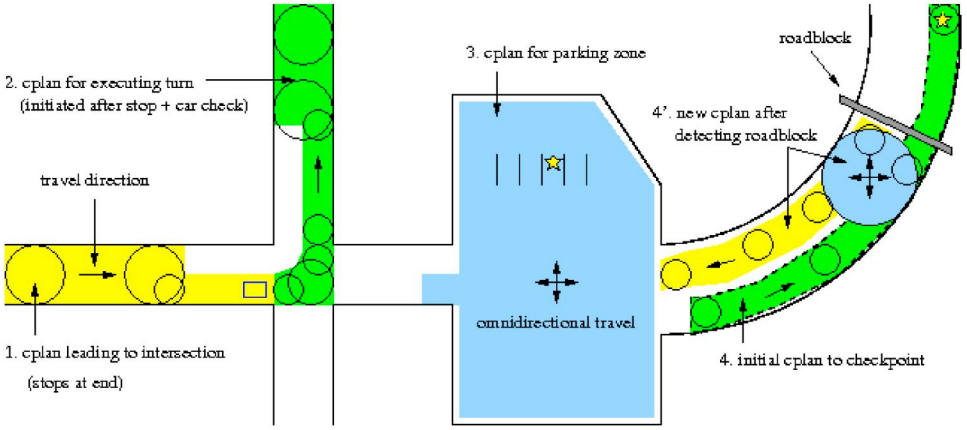

Figure 2: Corridor plan for three (static) driving scenarios: turning at an intersection, navigating in a parking lot and executing a U-turn.

The first corridor plan corresponds to driving down a road toward an intersection. Away from the intersection, travel in either lane is possible, with travel in the left hand lane for passing purposes. Near the intersection, only travel in the right hand lane is allowed. Each corridor segment includes an allowable direction of travel. Once the vehicle reaches the intersection and comes to a stop, the traffic planner checks to see if it is possible to execute the next stage of the mission plan. Once there is sufficient clearance in the intersection, a corridor plan through the intersection is generated. Again, each corridor segment includes an allowable direction of travel. In the parking zone, the corridor plan provides information on the region of allowable motion and indicates that motion in any direction is possible. A change in plan occurs when the a valid corridor plan cannot be found. Initially, a corridor plan is generated that takes the vehicle to the desired checkpoint (star). As the vehicle traverses the course, the corridor is determined to be blocked and the corridor plan fails. A new route is generated by the mission planner that indicates we must turn around and return from the direction we came. At this point, the traffic planner generates a new cplan that allows for omnidirectional motion in the middle of the street. The dynamic planner then computes a U-turn maneuver to satisfy the (new) mission goal.

Examples of additional scenarios that have been used in evaluating the system architecture and design choices include emergency maneuvers involving a vehicle approaching in our lane, approaching and passing a moving car, and a GPS outage while rounding corner, with state jump bigger than segment spacing.

Project Management

Figure 3 gives a high level view of our workplan. We break the project into five primary tasks, each associated with one of the teams in our organizational structure:

- Program Management (Integrated Product Team [IPT]) - overall management of all project activities, including setting and updating project specifications and goals, setting schedules, organizing design reviews (Rx), and managing funded and voluntary personnel

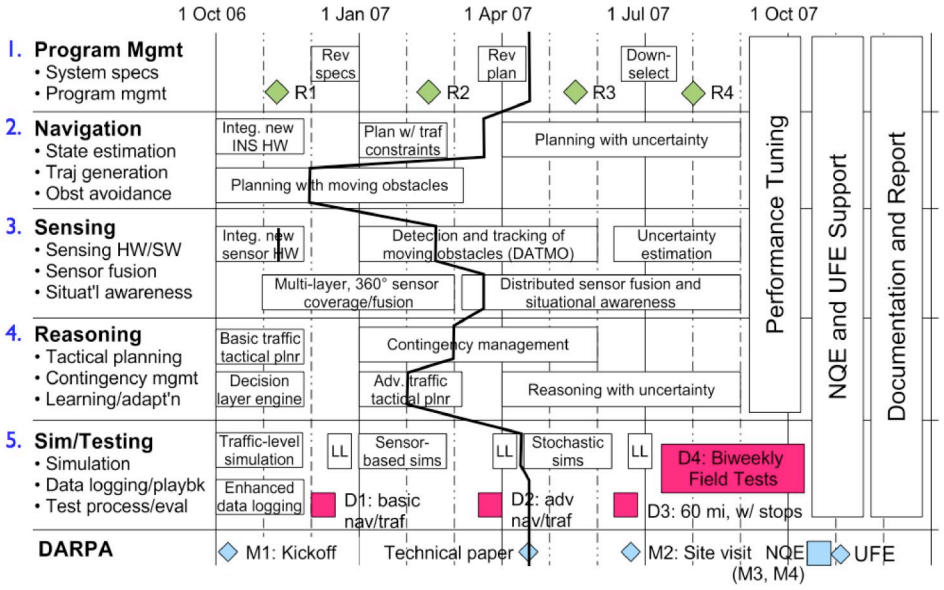

Figure 3: Task breakdown and milestone schedule. Rx = internal project reviews, Dx = internal demonstrations, Mx = DARPA milestones. The solid line indicates the current level of progress against each task.

- Navigation and Planning (Navigation Team) - transition and implement navigation algorithms developed by Caltech/JPL and Northrop Grumman Corporation (NGC). Develop, implement and test new algorithms for optimization-based planning in complex, dynamic, uncertain urban environments.
- Sensing and Mapping (Sensing team) - transition and implement sensing algorithms developed by Caltech/JPL under other funding. Develop, implement and test new algorithms for sensing, mapping in dynamic environments, and situational awareness.
- Mission and Contingency Management (Systems team) - transition and implement a systems architecture for mission and contingency management based on techniques developed at JPL that will allow robust operation in complex scenarios.
- Simulation and Testing (Systems team, Operations team) - simulation and testing of hardware and software associated with the project, as well as maintenance of vehicle infrastructure. This task will also be responsible for executing quarterly demonstrations (Dx).

2 Analysis and Design

In this section we describe the main subsystems in Alice and provide a summary of the technical approach we are taking, analysis results that led to design choices, and simulations and experiments to characterize component and subsystem performance.

2.1 Vehicle Hardware and Control Systems

Alice is a 2005 Ford E-350 van modified by Sportsmobile West to provide a rugged platform suitable for housing our computation, sensing and actuation systems (see [2] for a more detailed description). We have modified Alice with custom reinforced bumpers and roof sensor mounts

purchased from Aluminess Products. Alice's front and rear roof sensor mounts are fixed position slotted rail, which allows for quick and easy reconfiguration of the sensor package. Alice is currently outfitted with multiple fixed mount ladar sensors and stereo camera pairs, as well as an Applanix POS-LV unit for pose and filtered navigation information, as shown in Figure 4.

Figure 4: Team Caltech's autonomous vehicle, Alice. The left figure shows a front view of Alice, front which the horizontal ladars (embedded in the bumper), downward pointing ladars (above bumper and roof) and stereo cameras are visible. The right figure shows a side view, from which the same sensors and the Applanix GPS and DMI sensors are also visible.

By Summer 2007 we will have mounted an AC-20 vehicle tracking radar purchased from TRW, as well as at least one pan-tilt sensor platform.

A new power distribution system is being designed for Alice that replaces the previous collection of custom-built power distribution, with off the shelf solutions from the marine and RV industries. The entire electrical system is protected against interruption of the Honda EU3000 AC generator through two Accumentrics military-grade uninterruptable power supplies. These also serve as power conditioners and AC (80-265 V, 47-440 Hz) to 24 VDC converters. A multi-stage fusing and breaker design protects equipment from damage and maintains the safety of the vehicle.

We are purchasing an E-stop system from Torc Technologies as a drop-in replacement until the final installation of the DARPA-provided Omnitech Robotics E-stop. In addition to the remote-stop capability provided by this unit we are also able to stop the vehicle through one of 3 large kill-switches placed around the vehicle. One each is located on the rear-driver and rear-passenger panels, and the third is located within reach of the driver or passenger in the front of the cabin. The vehicle's E-stop is also fully triggerable through the computer system. Upon E-stop disable, the vehicle cuts power to the throttle as well as triggering a reserve pressure tank to fire and immediately fully depress the brake. E-stop disable can only be cleared by physically entering the vehicle and manually toggling the clear switch. Alice is also equipped with a tone and light system that operate continuously when the vehicle is running autonomously.

Vehicle pose The vehicle position, orientation and velocity are determined using an Applanix POS-LV 420 internal measurement system. The POS-LV is an advanced navigation unit that in-

tegrates two GPS antennas, an inertial measurement unit (IMU), differential GPS corrections and a wheel odometer. The POS-LV provides a full navigation solution that includes position in global coordinates (latitude, longitude and altitude), orientation along three axes (roll, pitch and heading), velocity and angular rates. Detailed information can be requested from the unit in the form of different messages, including estimated root mean square (RMS) precision values and precision ellipsoids, raw sensor data, and diagnostic information on the different subsystems.

We have run different tests to evaluate the performance of the Applanix POS-LV. We evaluated the estimate of the global position and of the vehicle kinematics by analyzing the RMS accuracy on the estimated variables. Such an accuracy provides a measure of the confidence interval around the estimated variable in which the real variable lies, and it is provided by the unit itself. Without a differential GPS receiver the latitude-longitude RMS accuracy usually remains greater than 2 meters. When the dGPS is connected and correctly working, the RMS precision is usually between 0.35 and 0.55 meters. The velocity vector components are estimated with an RMS accuracy of approximately 0.5 meters per second, while the orientation RMS accuracy is 0.015 degrees on each axis. We have also performed tests in which the GPS signal was blocked, because of tall buildings and trees covering the road. After a couple of miles driving, the POS-LV accuracy was still within a few meters, while the system used in Alice in the 2005 DARPA Grand Challenge was reporting an accuracy larger than 100 meters.

Trajectory Control The design specification for our trajectory tracking algorithm is to receive a trajectory from a planning module and output appropriate actuator commands to keep Alice on this trajectory. The inputs to the algorithm are the current state of the vehicle (position and orientation, along with first and second derivatives) and the desired trajectory (specified in northing and easting coordinates, with their first and second derivatives). From these inputs, the algorithm outputs steering and brake/throttle commands to Alice through a software interface. This capability was developed for the 2005 Grand Challenge and demonstrated the ability to provide +0/-10% for velocity tracking, and ±20 cm perpendicular y-error at 5 m/s, with larger errors allowable at higher speeds [2].

2.2 Sensing subsystem

We have developed approaches for sensing, mapping and situational awareness that build on past work at Caltech/JPL. The bottom layer of the sensing stack consists of the sensing hardware, and low level drivers and feeders that make the raw sensed data available to the perception algorithms. We have perception algorithms which detect and track lines on the road, static obstacles, traversable hazards, and moving vehicles. This data is then fed into the mapper and fused to form a map database which is used by the planners.

A primary focus of our work has been to effectively incorporate uncertainty and error compensation throughout the system. Uncertainty in the different perceptors can propagate through the sensing stack and is not only used for more effective multi-sensor fusion, but can be used by the planning subsystem to make safer and more informed control decisions. In addition to estimating the uncertainty in the positions of the sensed objects, we also include a metric of the confidence that the object exists. This confidence level is initially low, but increases as the object is tracked over time, and as more than one sensor detects the object. This allows us the flexibility to pass hypothetical objects that are sensed with less accurate, longer range sensors into the map even if we are not sure about whether the object really exists. This can be useful to provide the

planning subsystem with advanced warning that there may be a reason to slow down and to make sure our commanded velocity is safe in case the object does exist. As Alice gets closer to the hypothetical object, more sensors are brought to bear over longer time, and the uncertainty is reduced so that it is either removed from the map, or has a very high confidence of existence and we plan around it.

Another important design advantage for our sensing stack is the networked architecture that we have developed and tested on Alice. The advantage of using a networked architecture is the ability to leverage a large number of sensors in a modular fashion. However, this distributed approach also brings several new challenges, including managing communication resources and accounting for time skew and spatial registration between sensors. These challenges are met by our sensor feeders which enable efficient and effective registration and sharing of the large amounts of sensed data.

In this section we will give a more detailed overview of the sensing hardware, the sensor feeders, the line and obstacle detectors and trackers and the mapping module.

Sensing hardware The sensing hardware for Alice was chosen based on an assessment of possible causes of mission failure, comparison of the sensor converge against each of the technical evaluation criteria and the current sensor footprint of the vehicle. Figure 5 provides a summary of the current planned sensor suite for the vehicle. The use of a pan/tilt unit was chosen to provide the ability to make use of a long range, narrow beam RADAR at distances of up to 200 meters. This pan tilt unit also allows additional sensors to be directed toward the rear of the vehicle, when needed. An infrared camera is being evaluated as a mechanism to allow better identification of other vehicles at intersections and while driving.

Feeder On the software side, each sensor is managed by a sensor feeder module. Feeders have four main responsibilities: (1) To interface with the sensor hardware, including start-up and shutdown sequences, configuration management and health monitoring. (2) To maintain accurate sensor calibration, such as focal length (for cameras) and the sensor-to-vehicle coordinate transforms (all sensors). (3) To perform data pre-processing, such as image rectification and dense stereo ranging (cameras). (4) To tag sensor data with the corresponding vehicle pose data, such that all data can be placed in a common coordinate frame.

Data from feeders is transmitted to perceptors using a specialized high-bandwidth, low-latency communication package called SensNet. SensNet's connection model is many-to-many, such that each feeder can supply multiple perceptors and each perceptor can subscribe to multiple feeders. Perceptors can therefore draw on any combination of sensors and/or sensor modalities to accomplish their task (e.g., a road perceptor can use both forward-facing cameras and forward-facing ladars). SensNet will also choose an appropriate interprocess communication method based on the location of the communicating modules. For modules on the same physical machine, the method is shared memory; for modules on different machines, the method is Spread/TCP/Ethernet.

The feeder/perceptor architecture, as supported by SensNet, affords tremendous flexibility in the design and development of perceptual components. It supports modular, parallel development (in which both new feeders and new perceptors can be easily added) and allows program-

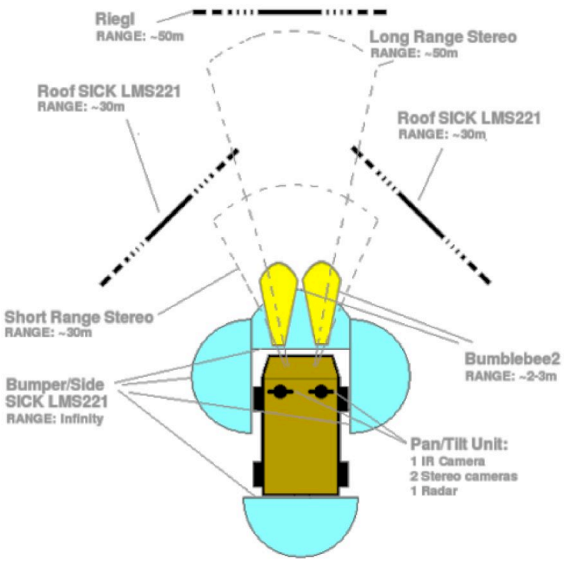

Figure 5: Sensor suite. Items marked with an asterisk have not yet been installed on the system.

mers to focus on algorithms rather than infrastructure. Latency remains a limiting factor, however: it takes up to 100 milliseconds to transmit a 2 MByte stereo image frame over Gigabit Ethernet, so perceptors that make heavy use of stereo image data must be co-located with their stereo feeders (where faster shared memory can be employed).

Figures 6 through 9 show example outputs from the sensor feeders. Figure 6 shows the left/right images from the roof-mounted long-baseline stereo camera pair, along with the range computed by the JPL stereo software (brighter is closer). Significant features such as cars, trucks and off-road obstacles are clearly discernible. Similarly Figure 7 shows the point cloud generated by the two roof-mounted terrain-sensing lidars (color-coded by elevation).

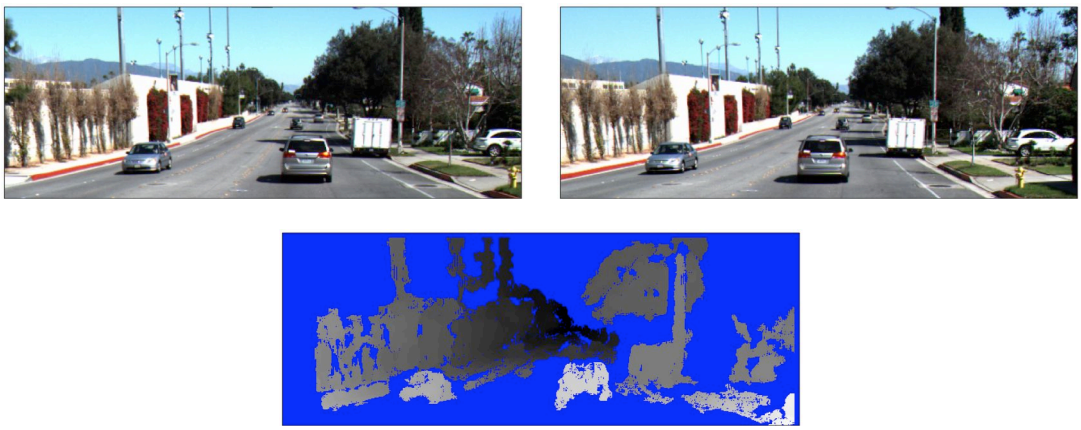

Figure 6: Output from the stereo feeder: left, right and range images (brighter is closer).

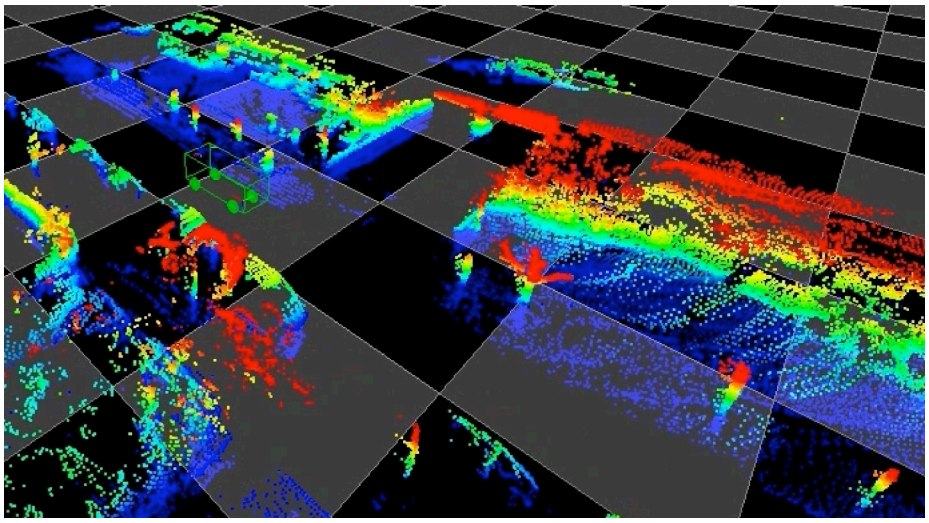

Figure 7: Output from roof ladar feeders (fused over multiple frames). The point clouds are colored by elevation, with blue low and red high.

Figure 8 shows data from multiple sensors (ladar and stereo) as visualized by our viewer utility. The data is first transformed into a common vehicle-centric frame, then projected into the world frame using pose data from the Applanix unit. The data is shown superimposed on orthorectified geo-references aerial imagery (for reference). Note the car crossing in front of Alice;

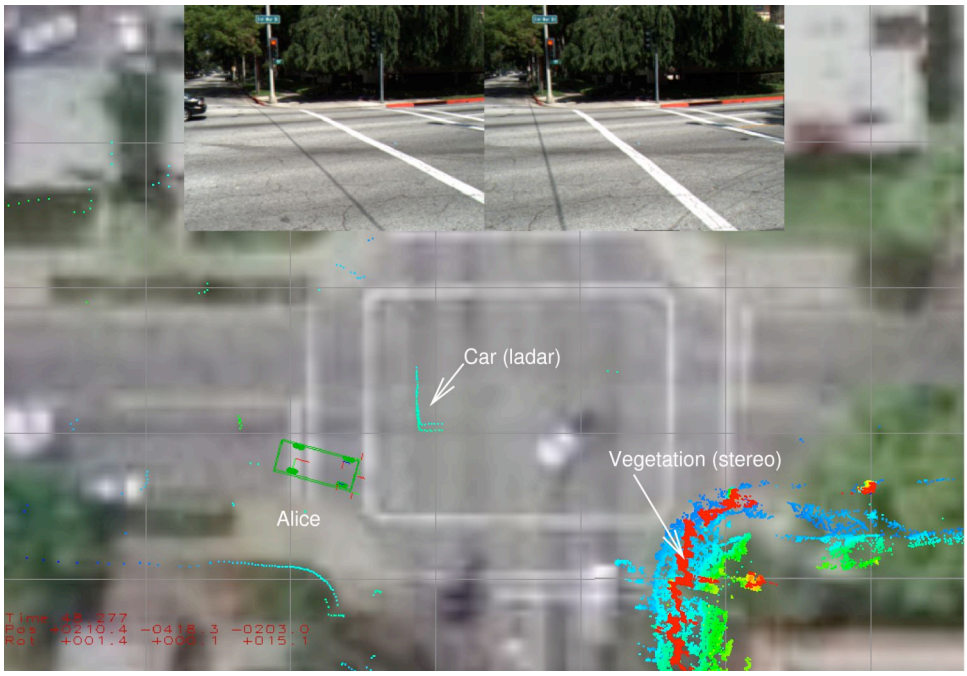

Figure 8: Combined stereo and ladar data for an intersection (the data is geo-registered and superimposed on a aerial image). Both ladar and stereo points are colored by elevation (blue low, red high). Note that the vehicle crossing in front of Alice is detected by both the side-facing and forward facing ladars.

this is not yet in the camera field-of-view, but shows up clearly in two of the ladar scanners. Since the ladars are not synchronized and the car is moving, the two scans are slightly different.

Figure 9 shows another intersection with oncoming traffic. In this case, the red car on the other side of the intersection is beyond the effective range of ladar, but appears quite distinctly in the stereo data.

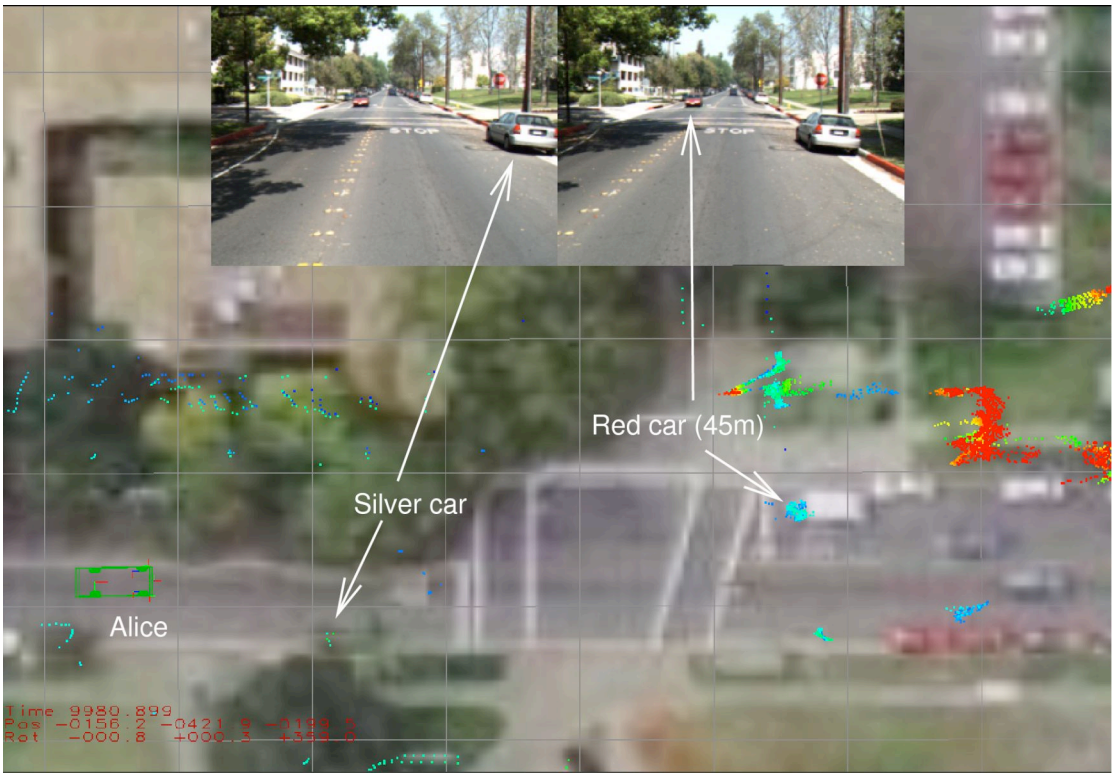

Figure 9: Combined stereo and ladar data for another intersection. The stereo detects the incoming red car on the other side of the intersection (approximately 45m), while the ladar detects the nearer (stationary) silver car.

Road Line Detection and Tracking We have developed methods to detect lane lines and stop lines from the image data collected by the short range stereo camera pairs. We first detect candidate lines in each frame, and then track multiple lines over time to further refine our estimate and reduce noise. The tracked lines are passed to the mapper where they are associated with prior RNDF line information to refine our situational awareness.

The first step of the algorithm is to remove the perspective effect from the image. This helps in solving the problem of varying width of line in the image. The technique used is called Inverse Perspective Mapping (IPM), in which we assume the road is a flat plane, and use the camera intrinsic (focal length and optical center) and extrinsic (pitch angle, yaw angle, and height above ground) parameters to take a top view of the road. This makes the width of any line uniform and independent of its position in the image, and only dependent on its real width in reality. It also removes the perspective effect, so that lines parallel to the optical axis will be parallel and vertical.

After taking a top view of the image, we use convolution kernels to detect near horizontal (or near vertical) lines in the image. By using a kernel of the appropriate shape, we can emphasize lines of specified width. For a horizontal line, the kernel has a Gaussian in the x-direction and a second-derivative of Gaussian in the y-direction. The scales of the horizontal and vertical aread-justed according to the width and height of the type of line to be detected. For example, the scale in the horizontal direction can be chosen such that we have enough width to detect at least 2 m lines (whose pixel equivalent can be computed from the IMP transformation). Similarly, the vertical scale is chosen to detect lines of thickness 12", allowing us to detect stop lines bounded by these dimensions.

Next, we threshold the image to get rid of the lower responses. This is done by keeping only the top 2% percentiles of the filtered image pixels, which correspond to the highest responses obtained from the stop line in the image. We then perform pixel grouping on the remaining pixels, to get the parameters of the line in the image (position and orientation). We used Hough Transform line grouping algorithm, which provides flexibility in detecting lines of any orientation or position in the image. The orientations are searched between ±10 degrees of horizontal or vertical, which allows for lane features that are not orthogonally aligned to the vehicle to be detected.

Finally, after detecting the position and orientation of the line, we work on determining the two endpoints of the line. This works by getting the pixels belonging to the detected stop line, using Bresenham's line drawing algorithm [1], and then convolving a smoothed version of the pixel values on the line with two kernels representing a rising and a falling edge, and getting the points of maximum response. The overall approach is summarized in Figure 10.

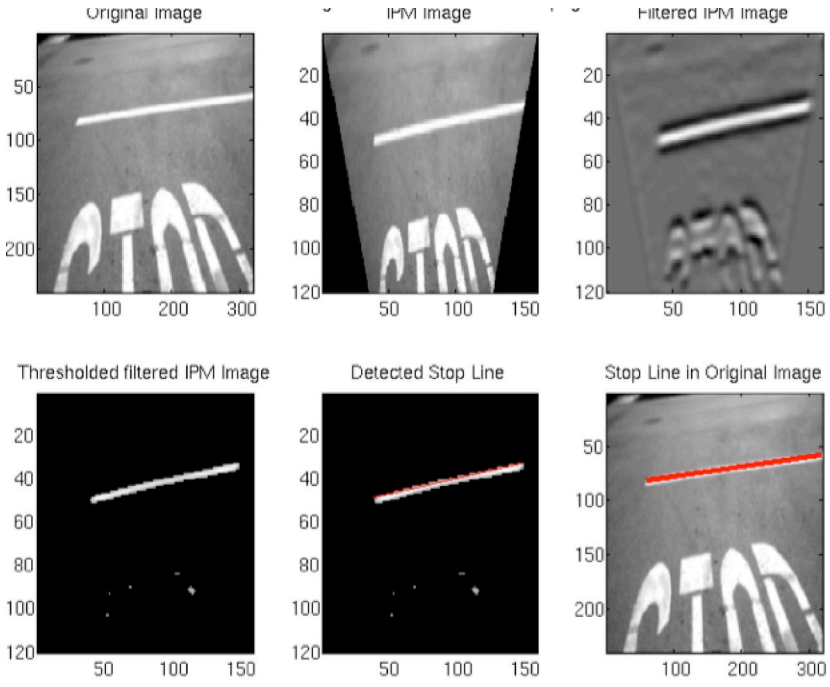

Figure 10: Summary of stop line algorithm.

After a line is detected, we need to choose a representation for the line to be able to track it over time. Choices include the coordinate frame for tracking the line (image, vehicle or ground

referenced), the representation of the stop line (midpoint or endpoints), and motion models (constant velocity, constant acceleration). Five different combinations have been chosen and tried out, which gave us five different models and implementations. After identifying the model, the representation, and the coordinate frame, a Kalman filter is implemented to track the stop lines detected. A Kalman filter tracker is initiated when we detect 3 stop lines in three consecutive frames that are sufficiently close to each other. This helps eliminate triggering on false positives in the detection algorithm. Then, if there are no detections for 5 consecutive frames, we eliminate the existing tracker, as this indicates that the stop line has gone out of the camera view.

Obstacle Detection and Tracking

Our primary sensors for object detection and tracking are the four bumper-mounted ladars. In an urban environment most obstacles that are of concern, in particular other vehicles, will be at least bumper height and approximately vertical. Thus, a 2-dimensional representation of the world for the purposes of object tracking is a suitable first approximation. Future options include adding perceptors that take data from the two sweeping ladars (for terrain information), multiple stereo pairs and a radar unit for long-range moving object detection and tracking.

Currently, ladar range and angle data is received from any one of the four bumper-mounted ladar units, as well as the current state estimate for the vehicle pose. The output of this perceptor is a list of tracked objects, both static and dynamic, sent to the mapper. The architecture has been divided into discrete steps: segmentation, initial data association, classification, model update and cleanup.

The first step is to identify individual objects from each input ladar scan. Discontinuities in adjacent range measurements above a threshold are used to break the scan up into objects. The points for each object are immediately projected into the local coordinate system, relying on the accuracy of the state data and sensor calibration. All further calculations are performed in this frame, allowing the algorithm's assumptions to be independent of Alice's current motion.

Due to the 75 Hz update rate of the ladar units and the relatively slow speed of vehicles in the race, a good initial association of new data to previously seen objects can be made simply using proximity. Once new data has been associated with the object, the Kalman filter estimating position and velocity is updated, but only if it is possible to match an edge of the object to a previously seen edge. Here, occlusion is taken into account to differentiate between one edge of an object truly moving and an object appearing to grow as it is revealed by another moving object in the foreground. If after a certain number of observations the object's estimated velocity is above a threshold, it is classified as a car. Cars are represented by a rectangle, which allows more precise tracking of features of the car through partial occlusions.

This system has been shown to work well at tracking objects in a simple environment, including correctly tracking cars through complete occlusions. Figure 11 shows some of the current capabilities of this perceptor.

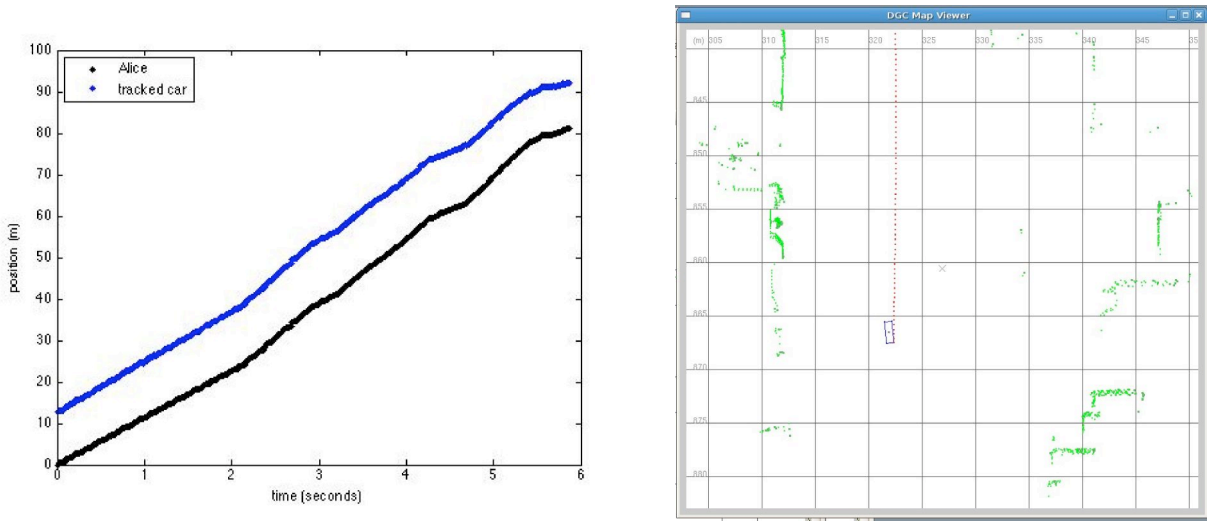

Figure 11: Moving obstacle detection. The left figure shows the estimated position of Alice and a vehicle moving in front of Alice under manual driving. The same scenarios is shown on the right, with additional static obstacles present. The green dots are ladar returns that have been classified as belonging to static objects, the red dots are the previous path of the tracked car, and the blue box is the tracked car.

Mapper The mapper defines the interface between the sensing and the planning subsystems. The mapper receives tracked map elements from the various sensors and fuses this sensed information together to maintain a cohesive map element database. The database consists of static and dynamic sensed obstacles, as well as sensed road features and lines. Map elements are represented as 2.5 dimensional vectorized objects with a representation of uncertainty in position, as well as uncertainty in velocity and height if applicable. The mapper associates the sensed road features with the RNDF prior information, and this method can be extended to become the basis for sensor based localization if needed. Each map element also maintains a notion of the confidence of existence which is determined by the quality of the detection, the amount of time the element has been tracked, and the number of different sensors which detect the element.

We are in the early stages of fusing objects detected by both the stereo cameras and the ladars. Figure 12 shows our 3D viewer visualizing the ladar data taken at the same time as the stereo data. Figure 12b shows Alice in white and the vehicle position estimates in yellow projected into the planar map. The red boxes represent tracked static obstacles. The figures shown represent data taken at a single time stamp and are meant to give a brief example of fusing data and populating the map.

The mapper not only maintains the database of obstacles and road features, but it also associates obstacles with elements of the road topology. For example, the planning subsystem can query the map for all obstacles in a single lane or for all obstacles in the range of sensor visibility for a specific intersection. The mapper also maintains an estimate of the cost of traversal for lengths of road, which it passes to the mission planner to help refine the graph search for the best route to complete the mission.

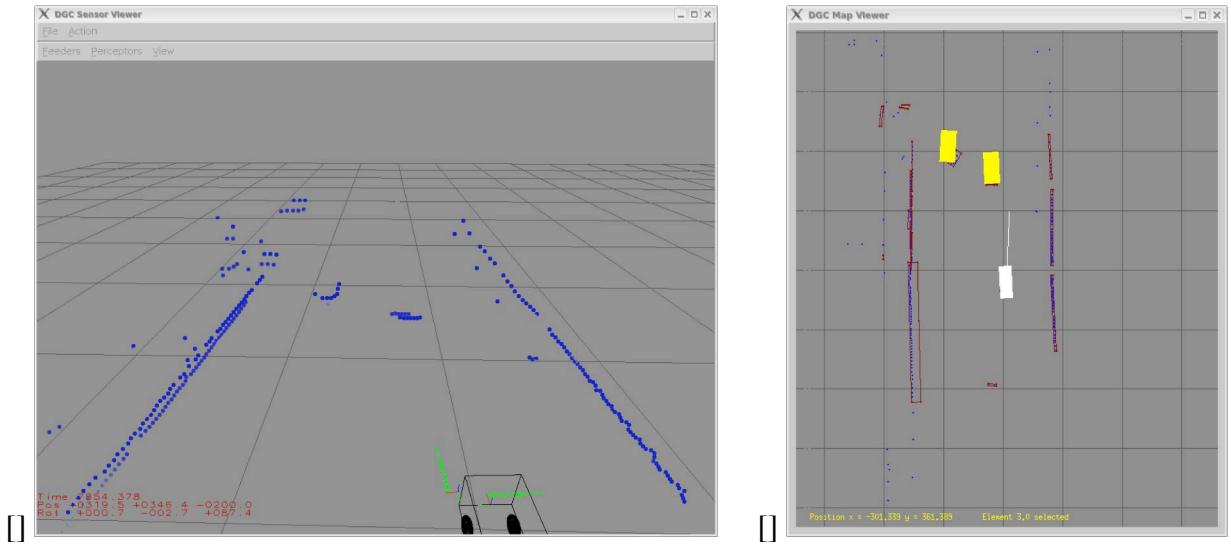

Figure 12: Map representation with Alice in white and two tracked vehicles in yellow. (a) Ladar and stereo vision data from an experimental run. (b) Map representation of the same scene.

2.3 Planning subsystem

Planning in urban environments is challenging for a number of reasons: first the environment is generally structured and, combined with traffic rules, specific vehicle behavior is required. Second, the environment is dynamic, including other autonomous vehicles. Third, there will always be uncertainty in the sensed environment and the plans have to reflect this. To accomplish this planning task, a multi-level decomposition as shown in Figure 1 is used. A mission planner determines a route through the road network (specified in the RNDF) based on the current mission (from the MDF). The route plan is modified to account for blockages and other known traffic conditions, and is the input to a traffic planner. The traffic planner combines this route (goal) with the sensed static environment and traffic rules to determine what motion is allowed. It reasons abstractly about other vehicle behavior to adjust the plan and provides a corridor description to the optimization-based path planner. The corridor contains geometric constraint information, as well as a cost map and velocity profile. The path planner integrates the corridor information with dynamic obstacle motion prediction to determine a dynamically feasible path that avoids obstacles and follows traffic rules. This planner is also capable of planning in unstructured zones.

In this section we describe the individual algorithms that we have developed using this decomposition, beginning with the software architecture used to implement the planning subsystem.

Canonical Software Architecture The modules that make up the planning system are responsible for reasoning at different levels of abstraction. Hence the planning system is decomposed into a hierarchical framework. To support this decomposition and separation of functionality, while maintaining communication and contingency management, we implement the planning subsystem in a canonical software architecture (CSA) as shown in Figure 13.

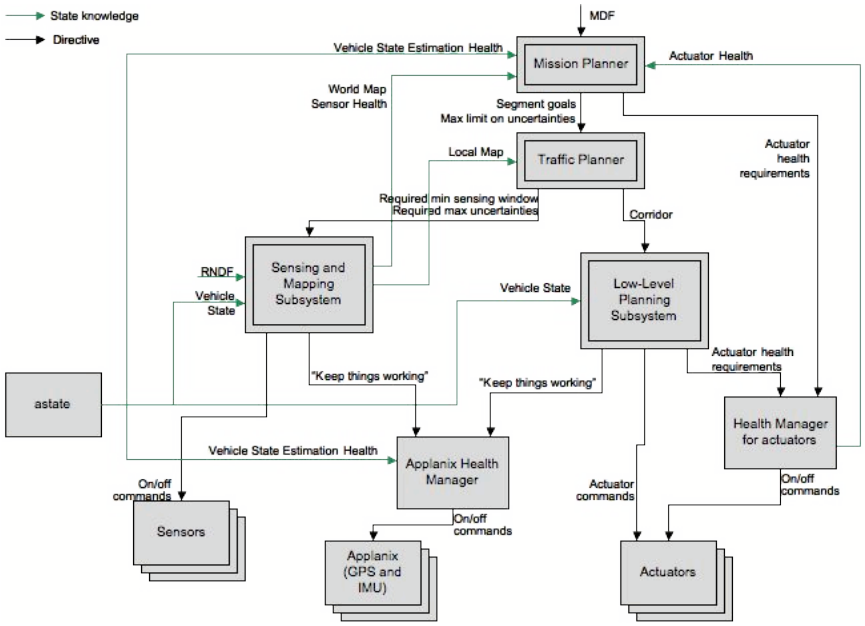

Figure 13: The planning subsystem in the Canonical Software Architecture. Boxes with double lined borders are subsystems that will be broken up into multiple CSA modules.

This architecture builds on the state analysis framework developed at JPL [3, 8, 5] and takes the approach of clearly delineating state estimation and control determination. To prevent the modules from getting out of sync because of the inconsistency in state knowledge, we require that there is only one source of state knowledge although it may be captured in different abstractions for different modules.

For modularity, each planner may be broken down into multiple CSA modules. A CSA module consists of three components—Arbitration, Control and Tactics—and communicates with its neighbors through directives and status messages, as shown in Figure 14.

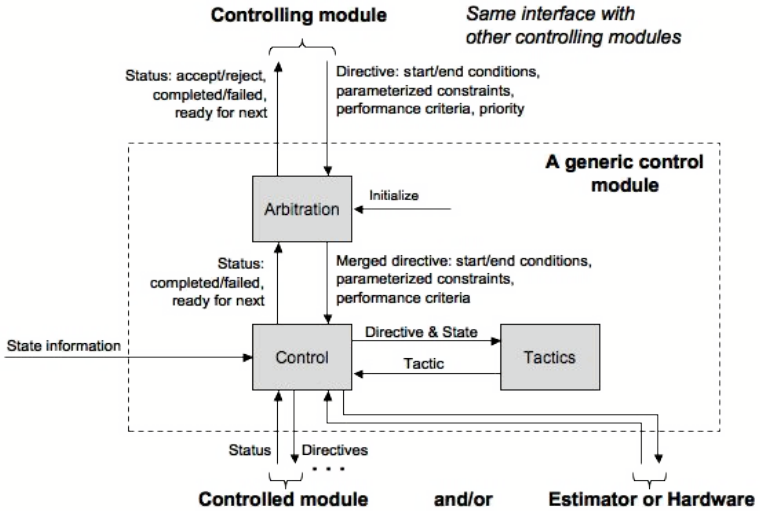

Figure 14: A generic control module in the Canonical Software Architecture.

Arbitration is responsible for (1) managing the overall behavior of the module by issuing a merged directive, computed from all the received directives, to the Control; and (2) reporting failure, rejection, acceptance and completeness of a received directive to the Control of the issuing module. Control is responsible for (1) computing the output directives to the controlled module(s) based on the merged directive, received status and state information; and (2) reporting failure and completeness of a merged directive to the Arbiter. Tactics provides the core functionality of the module and is responsible for generating a control tactic or a contiguous series of control tactics, as requested by Control.

Mission Planner The mission planner has five main responsibilities and is broken up into two estimation and three CSA control modules: the estimation of the traversibility graph which represents the connectivity of the route network (Traversibility Graph Estimator), the estimation of the vehicle capability which is an abstraction of the system health (Vehicle Capability Estimator), the determination of mission goals and conditions under which we can continue the race (Mission Control), the determination of segment-level goals (Route Planner) and the management of system health (Vehicle Capability Control).

The route planner communicates with the traffic planner using the common CSA interface protocols. Thus, it will be notified by the traffic planner when a segment-level goal directive is rejected, accepted, completed or failed. For example, since one of the rules specified in a segment-level goal directive is to avoid obstacles, when a road is blocked the directive will fail. Since the default behavior of the traffic planner is to keep the vehicle at pause, the vehicle will stay at pause while the route planner replans the route. When the failure of a segment-level goal directive is received, the route planner will request an updated traversibility graph from the traversibility graph estimator module. Since this graph is built from the same map used by the traffic planner, the obstacle that blocks the road will also show up in the traversibility graph, resulting in the removal of all the edges corresponding to going forward, leaving only the U-turn edges from the current position node. Thus, the new segment-level goal directive computed by the Control of the route planner will be making a U-turn and following all the U-turn rules. This directive will go down the planning hierarchy and get refined to the point where the corresponding actuators are commanded to make a legal U-turn.

Traffic Planner The Traffic Planner is a software subsystem that encapsulates three main responsibilities: the estimation of the autonomous vehicle's state in the context of a traffic situation (Traffic State Estimator), the determination of a control action based on the estimated traffic state, mission goals and sensing data (Traffic Planner Control) and the determination of a set of hard constraints that define the boundaries of a safe navigable corridor for the autonomous vehicle based on control action and sensing data (Corridor Determination). Three CSA modules and their interaction are shown in Figure 15.

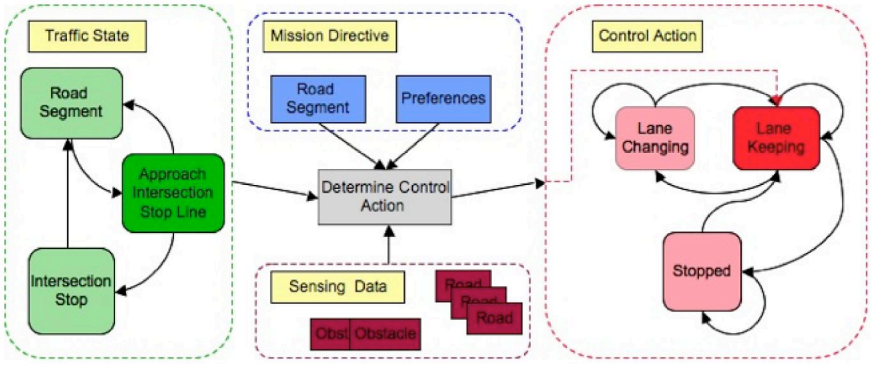

Figure 15: Control Action Determination by the Traffic Planner Control subsystem. Note: Traffic State and Control Action finite state machines pictured here are not complete.

The Traffic State Estimator estimates traffic state, which is an abstract representation of the autonomous vehicle's current traffic situation. This traffic state is constructed and determined by simultaneously considering sensing data and the vehicle's state information (position and velocity) and comparing it to a known list of possible traffic states. The Traffic Planner Control is responsible for making high-level control decisions such as "Lane Keeping" and "Lane Changing". The control action is determined by evaluating a current segment-level goal while concurrently assessing both the traffic state estimate and relevant real-time sensing information. The Corridor Determination calculates a safe corridor and velocity profile based on a determined control action and traffic state. The resulting corridor is analyzed for feasibility and safety with respect to potential conflicts such as blocking static obstacles or potential collision with other dynamic obstacles.

Traffic states and control actions are encoded as two separate probabilistic finite state machines. The motivation for this separation is to delineate traffic state knowledge from control and corridor determination, thereby reducing dependencies that could lead to the degeneration of system robustness. In addition, such a design promotes a clearer separation of functionality that in turn is amenable to extension and better code organization.

The probabilistic finite state machine is implemented as a directed graph whose nodes are states, corresponding to either a traffic state or a control action, and whose directed edges define transition conditions between either traffic states or control actions.

To reduce the likelihood of state explosion, we determined a small set of traffic states and control actions that respectively describe traffic situations and driving criteria derived from DARPA guidelines. Our intention is to describe traffic states in a specific yet reusable way in order to subsequently specialize them with respect to dynamic traffic situation that are relevant for control action determination. For example, a "Road Segment" traffic state may be specialized with information on obstacles in the neighboring lane if we are determining whether the control transition to "Lane Change" is feasible. On the other hand, the "Intersection Stop" traffic state may be specialized with information on obstacles at other parts of the intersection, along with precedence rules, to determine whether the control transition "Stopped" to "Lane Keeping" is allowed (i.e. if it is possible to proceed through the intersection).

During traffic state estimation, a set of traffic states is computed with a measure of uncertainty by evaluating transition conditions of the current traffic state with respect to the autonomous vehicle's position/velocity and sensing data. All transitions between states encapsulate the set of conditions by which a traffic state is reached. During the evaluation of such transitions, the Traffic State Estimator determines the probability associated to meeting each transition's particular conditions. A probability distribution is computed on the set of possible traffic states and the most likely state is determined.

The Traffic Planner Control determines the control action by simultaneously evaluating rules associated to the composition of estimated traffic state, the mission directive, sensing data and mission preferences. These rules along with associated traffic rules are encapsulated in control transitions. The Traffic Planner Control calculates the probability associated with meeting a control transition condition by augmenting it with a measure of safety and a qualitative assessment describing the reason and degree to which a control transition has not been met.

Corridor determination is implemented generically for each control action, and specialized at the time of control action determination. The assumption is that each control action has a generic corridor associated to it but that it changes with respect to parameterizations. For example the "Lane Changing" corridor has a general shape but is parameterized by the lane to change into. The velocity profile is determined after the corridor is determined and is constrained by traffic rules (speed limits).

Dynamic planner Alice's previous architecture used a speed-based cost map to determine how to drive through a given section of terrain as quickly as possible. To extend our optimization-based controller to handle dynamic environments, we are making use of recent developments in optimization-based path planning at Northrop Grumman and Caltech, which have been implemented in the OTGX (optimal trajectory generation) software package [4]. OTGX extends previous work at Caltech in real-time trajectory generation [6, 7] by using non-uniform rational B-splines (NURBS) as a basis for exploring optimal paths that satisfy the dynamics of the vehicle as well as constraints on inputs and states.

The dynamic planner solves an optimization problem specified by the segment goal, the corridor plan and local features, including the moving vehicles and road features. The cost function for the dynamic planner is in the form

$$J = \int_{t_0}^{t_0+T} \left(L_s(x,\tau) + L_m(x,\tau) + L_g(x,\tau) \right) d\tau + V_g(x(T),T) \qquad (1)$$

where

$L_s(x,\tau)$ = static cost due to terrain, corridor, static obstacles

$L_m(x,\tau)$ = dynamic cost due to moving vehicles

$L_g(x,\tau)$ = integral cost associated with mission goal (optional)

$V_g(x(T),T)$ = terminal cost associated with mission goal

The terminal time T is left free.

Figure 16 illustrates the generation of trajectories using the OTGX package.

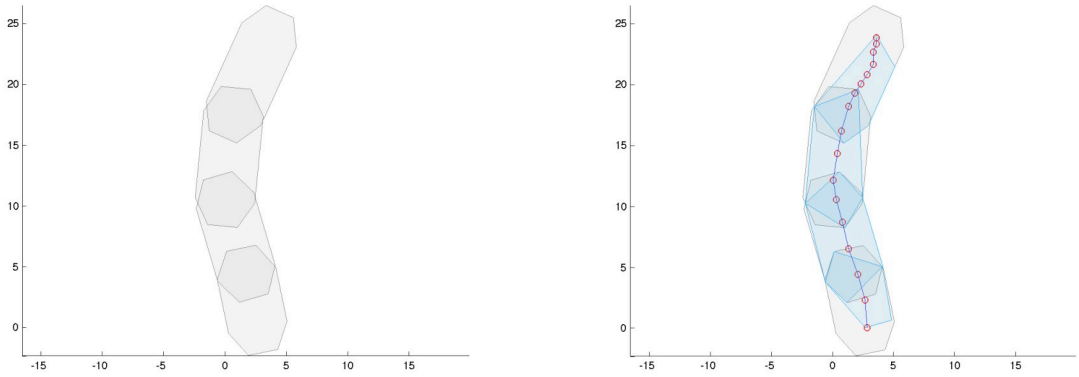

Figure 16: Dynamic planner. The figure on the left shows a sample corridor passed to the dynamic planner. A set of overlapping polytopes is fit to this corridor for the purpose of choosing NURBS basis functions. These basis functions are then used to plan an optimal path that stays inside the corridor.

The development of core functionality in this module has lagged behind the planned schedule and has caused some delay in testing functions that require non-trivial planning computations.

2.4 Vehicle computing

Computing will provided using a a compact PCI computing system using 8 Intel Core 2 Duo and 2 Core Duo blades. This configuration provides several benefits, including high speed processing, efficient form factor and reduced power consumption. Each of the 10 blades will be connected to a common backplane with two 24-port gigabit ethernet switchboards.

2.5 Simulation capabilities

Team Caltech's core simulation software is composed of a kinematic simulator, which replaces the vehicle, and a traffic simulator, which replaces the environment. By running both of these programs, a developer can close the loop around the planning stack and test a variety of scenarios like those that will be encountered in the race. The simulation will soon be capable of simulating almost all scenarios that might be encountered at the site visit.

The kinematic simulator uses a kinematic model of Alice to update the vehicle state at each simulation time step. This information can then be sent to other software modules that request it. Noise is optionally added to this state data, increasing the realism of the simulation.

The traffic simulator is primarily intended to test navigation and traffic-planning software. It offers a flexible combination of pre-designed, scenario-specific tests that can be loaded as modules, and interactive, on-the-fly additions such as the ability to add vehicles and roads at runtime. The interactive GUI is shown in Figure 17.

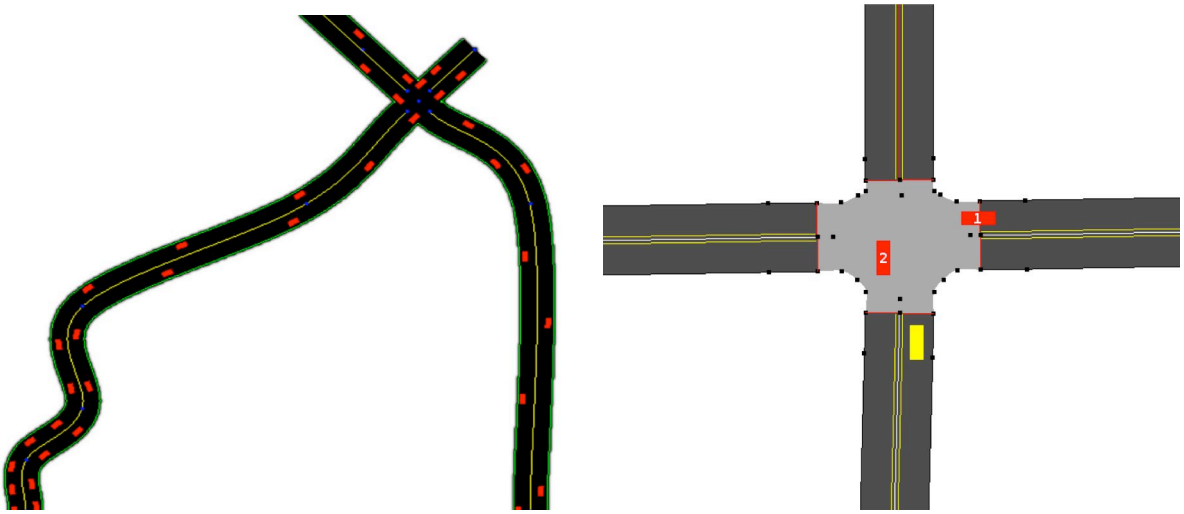

Figure 17: A four-way intersection in the traffic simulator. The simulated cars are red, and Alice is yellow. Cars 1 and 2 approached the intersection at almost the same time, so Car 1 has stopped and is waiting for 2 to clear the intersection before proceeding.

The traffic simulation software can create road networks with intersections and arbitrary number of lanes. These can either be created interactively, or by loading from an RNDF or configuration file. It is capable of simulating both individual cars and continuous traffic streams. Like roads, these cars can either be added during simulator or preloaded. The vehicles all follow basic traffic rules, such as avoiding collisions with other vehicles (including "Alice") and obeying intersection rules. Stopped vehicles can be added in arbitrary locations, and moving vehicles are added to specific lanes and follow a set path than can be modified as desired.

In addition to autonomous vehicles, the traffic simulator can load an externally-controlled vehicle, commanded either by another software module or by the user via keyboard commands. In this way, "Alice" is simulated in the simulation environment and can interact with the other vehicles. Once an Alice vehicle is added to the traffic simulator, the software can then extract environment information around Alice and transmit it to the mapping software, thus mimicking the vehicle sensors. In addition to these dynamic objects, static obstacles of arbitrary size can be preloaded or placed anywhere in the environment with the click of a mouse. The traffic simulator can save and load complete environment configurations, allowing the user to run specific tests repeatedly.

3 Results and Performance

In this section we describe some of results obtained thus far using our system, including the report of the independent test team.

Substantial work has been done on demonstrating the ability of the system components to perform their desired function using the simulation environment described above. An example scenario is shown in Figure 18.

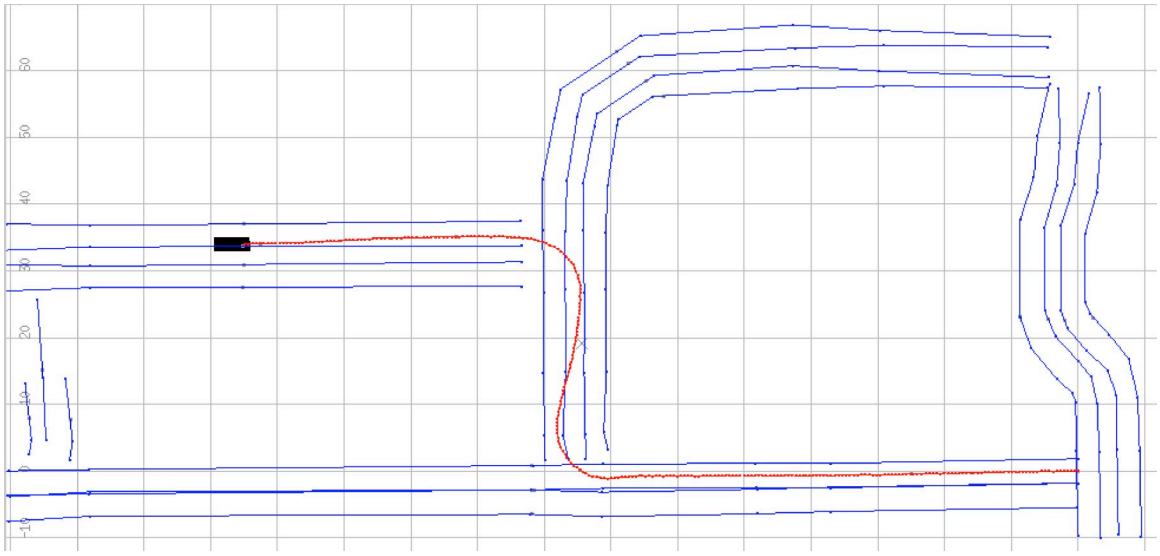

Figure 18: Sample simulation of planner stack.

Additional simulations have been run using hand placed obstacles and we have demonstrated the ability of the mission and traffic planners detect a blocked road, replan the route and execute a U-turn. Due to limitations in the capabilities of the dynamic planner at the time of testing, it was not possible to run this scenario on the vehicle.

Experimental demonstrations and evaluations have also been carried out and we have found that our simulation environment provides a faithful description of the system dynamics. In particular, we see relatively little different in the performance of the planning subsystem between the simulation and the full vehicle. The sensing subsystem can not be fully tested in closed loop operation through simulation, but extensive testing on logged data has demonstrated the efficacy of the modules that have been developed, as summarized already in Section 2.2.

3.1 Independent Test Team Report

Three sets of field tests have been planned by the Caltech team. Scheduled for December 2006, March 2007 and June 2007, these tests are based on the capabilities required by the DARPA Technical Evaluation Criteria. The first field trial covered the technical evaluation criteria specified in sections A1–A12 and C5. The second field trial covered the technical evaluation criteria specified in sections B1–B4 and C1–C6. The final field trial scheduled for June will cover the technical evaluation criteria specified in sections D1–D9. In addition the final trial will have a 60 mile mission level test. Beyond the field trials, during the Summer of 2007 a series of bi-weekly tests will be conducted to verify additional capabilities and search for operational weaknesses.

The Independent Test Team (ITT) has developed a set of test suites which cover four different areas: Basic Navigation, Traffic Interaction and Obstacle Avoidance, Traffic Law Compliance, and Mission Level Testing. Individual tests or sets of tests from each test suite are selected to verify that a particular element of the technical evaluation criteria is satisfied. All tests developed by the ITT consist of a RNDF file and MDF file based at one of the Caltech team test sites. To date these have been at the St. Luke's test site, as shown in Figure 19.

Figure 19: St. Luke test area

The ITT does not have prior knowledge of the tests other than the technical evaluation criteria that will be evaluated.

On December 17, 2006 the first of a series of tests performed by the ITT was completed. Alice attempted to execute 10 of the 16 tests which were available. Of the 10 attempted all but 2 executed to completion. Of the 8 that executed to completion, none were an unqualified success (see Figure 20).

Test Description	Attempted	Completed	Result
B1.1 - Starting Outside of road network	Yes	No	Failed
B1.2 - Start on road network	Yes	Yes	Passed, Provisionally
B1.3 - Stop mission and restart	Yes	Yes	Passed, Provisionally
B1.4 - Stop mission, move vehicle, restart	Yes	Yes	Passed, Provisionally
B2.1 - Stay in lane straight line driving	Yes	Yes	Passed, Provisionally
B3.1 - Stay in lane curved path driving	Yes	Yes	Failed
B5.1 - Get into proper lane while turning at intersection	Yes	Provisionally	Failed
B9.1 - Avoid obstacles while staying in lane	Data collection only	No	Failed
B4.1 - Stop at stop line, continue straight.	Yes	Yes	Passed, Provisionally
B4.2 - Stop at stop line, turn left.	Yes	No	Failed
B4.3 - Stop at stop line, turn right.	Yes	Yes	Passed, Provisionally
B9.2 - Leave lane to avoid a stationary obstacle	No	No	Failed
B9.3 - Leave lane to pass a moving vehicle (not in December)	No	No	Failed
B7.1 - U-turn at a designated exit/entry point	No	No	Failed
B7.2 - U-turn for an obstacle not at a designated exit/entry point	No	No	Failed
B3.2 - Stay in lane curved path driving sparse waypoints	No	No	Failed

Figure 20: Field test 1 results.

The December tests were based on code derived from the previous Grand Challenge, and at that point, none of the sensors other than GPS were operational. However, no tests that required obstacle detection were attempted. The vehicle did not yet have the capability to reverse, so all tests that required a U-turn were not attempted.

On March 18, 2007 the second of a series of tests performed by the ITT was completed. Alice attempted to execute 12 of the 27 tests which were available. Of the 12 attempted all but 3 executed to completion. Of the 9 that executed to completion, only one failed, and three were conditionally successful (see Figure 21).

Test Description	Attempted	Completed	Result
B1.1 - Starting Outside of road network	Yes	Yes	Passed
B1.2 - Start on road network	Yes	Yes	Passed, Provisionally
B1.3 - Stop mission and restart	Yes	Yes	Passed
B1.4 - Stop mission, move vehicle, restart	Yes	Yes	Passed
B2.1 - Stay in lane straight line driving	Yes	Yes	Passed, Provisionally
B3.1 - Stay in lane curved path driving	Yes	Yes	Passed, Provisionally
B5.1 - Get into proper lane while turning at intersection	Yes	No	Failed
B9.1 - Avoid obstacles while staying in lane	Yes	No	Failed
B4.1 - Stop at stop line, continue straight.	Yes	Yes	Passed
B4.2 - Stop at stop line, turn left.	Yes	Yes	Failed
B4.3 - Stop at stop line, turn right.	Yes	Yes	Passed, Provisionally
B9.2 - Leave lane to avoid a stationary obstacle	No	No	Failed
B9.3 - Leave lane to pass a moving vehicle (not in December)	No	No	Failed
B7.1 - U-turn at a designated exit/entry point	No	No	Failed
B7.2 - U-turn for an obstacle not at a designated exit/entry point	No	No	Failed
B3.2 - Stay in lane curved path driving sparse waypoints	No	No	Failed
M1.1 - Mission Level Test	Yes	No	Failed
New tests for March:			
T1.1 - Precedence	No	No	Failed
T2.1 - Following Traffic	No	No	Failed
T1.2 - Queueing	No	No	Failed
T3.1 - Drive Through Lot with Stationary Obstacles	No	No	Failed
T3.2 - Drive Through Lot with Moving Obstacles	No	No	Failed
T3.3 - Park in Designated Space in Empty Lot	No	No	Failed
T3.4 - Park in Designated Space, One Adjacent Space Occupied	No	No	Failed
T3.5 - Park in Designated Space, Both Adjacent Spaces Occupied	No	No	Failed
B7.2 - U-turn for an obstacle not at a designated exit/entry point	No	No	Failed
B3.2 - Stay in lane curved path driving sparse waypoints	No	No	Failed

Figure 21: Field test 2 results.

Although the March test was designed as a test of traffic interaction, it essentially became a retest of the December field test. The software had been almost completely replaced with a new architecture that supported traffic interaction and road following. However, the new code did not support much more in terms of capability than was available in December. The March tests showed that the new software was better than that available in December, but few new capabilities were ready. Sensing was tested on a stationary vehicle but the ability to avoid the obstacle was not yet implemented. In this case Alice stopped at the obstacle and attempted a U-turn. The U-turn function was not working correctly, so all U-turn tests were skipped as were other vehicle interaction test. A retest of the March field test is planned.

3.2 Summary and Current Status

In the past 6 months, Team Caltech has developed and begun to implement and demonstrate an autonomous systems architecture that is capable of performing the tasks required for the urban challenge. Substantial work remains to be done, both in terms of the performance and robustness of individual modules, and the overall system integration and testing. Slow progress on the dynamic planner has delayed our ability to perform some tests at the desired level, but the other system functions are progressing rapidly. To help speed development, an alternative planner is being developed that will use a simplified approach for planning in corridors capable of handling many of the simpler traffic situations.

Acknowledgements

The research in this paper was supported in part by the Defense Advanced Research Projects Agency (DARPA) under contract HR0011-06-C-0146, the California Institute of Technology, Big Dog Ventures, Northrop Grumman Corporation, Mohr Davidow Ventures and Applanix Inc.

The authors would also like to thank the following members of Team Caltech who contributed to the work described here: Daniel Alvarez, Brandt Belson, Julia Braman, William David Carrillo, Arthur Chang, Edward Chen, Steve Chien, Jay Conrod, Iain Cranston, Lars Cremean, Josh Doubleday, Tom Duong, Luke Durant, Josh Feingold, Matthew Feldman, Tony and Sandie Fender, Nicholas Fette, Ken Fisher, Brent Goldman, Scott Goodfriend, Steven Gray, Rob Grogan, Jerry He, Mitch Ingham, Michael Kaye, Aditya Khosla, Ghyrn Loveness, Russell Newman, Noele Norris, Eloka Ochuba, Kenny Oslund, Jimmy Paulos, Bob Rasumussen, Christopher Rasumussen, Chris Schantz, Chess Stetson, Klimka Szwaykowska, Daniel Talancon, Daniele Tamino, Abhishek Tiwari, Pete Trautman, David Trotz, Glenn Wagner, Yi Wang, David Waylonis, Albert Wu, Francisco Zabala and Johnny Zhang.

References

[1] Jack Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[2] L. B. Cremean, T. B. Foote, J. H. Gillula, G. H. Hines, D. Kogan, K. L. Kriechbaum, J. C. Lamb, J. Leibs, L. Lindzey, C. E. Rasmussen, A. D. Stewart, J. W. Burdick, and R. M. Murray. Alice: An information-rich autonomous vehicle for high-speed desert navigation. *Journal of Field Robotics*, 2006. To appear.

[3] D. Dvorak, R. D. Rasmussen, G. Reeves, and A. Sacks. Software architecture themes in jpl's mission data system. In *Proceedings of 2000 IEEE Aerospace Conference*, 2000.

[4] M. E. Flores and M. B. Milam. Trajectory generation for differentially flat systems via NURBS basis functions with obstacle avoidance. In *Proc. American Control Conference*, 2006.

[5] M. Ingham, R. Rasmussen, M. Bennett, and A. Moncada. Engineering complex embedded systems with state analysis and the mission data system. *J. Aerospace Computing, Information and Communication*, 2, 2005.

[6] M. B. Milam. *Real-Time Optimal Trajectory Generation for Constrained Dynamical Systems*. PhD thesis, California Institute of Technology, 2003.

[7] M. B. Milam, R. Franz, J. E. Hauser, and R. M. Murray. Receding horizon control of a vectored thrust flight experiment. *IEE Proceedings on Control Theory and Applications*, 152(3):340–348, 2005.

[8] R. D. Rasmussen. Goal based fault tolerance for space systems using the mission data system. In *Proceedings of the 2001 IEEE Aerospace Conference*, 2001.